NYU 上海 SHANGHAI 纽约大学

COMPUTER SCIENCE
&
DATA SCIENCE

CAPSTONE REPORT - SPRING 2022

Complexity and Demonstration of Quantum Supremacy Algorithms

Robin Luo, Siyuan Yu

supervised by
Tim Byrnes & Olivier Marin

Preface

Quantum computing has garnered immense hype in the field of computer science and quantum physics. In 2019, Google's AI Quantum Computing Lab made groundbreaking research showing proof of quantum supremacy – meaning quantum computers can compute algorithms no classical computer can. Since this research, studies in this field have been promoted globally and its area of research, accelerated.
This capstone is made to introduce readers with little to no knowledge about quantum computing on how to demonstrate quantum supremacy and its complexity using random quantum circuits, IBMQ, and Qiskit. We hope this capstone will provide a direction and incite curiosity amongst those who want to get started understanding this ever-growing field.
Robin Luo is a Computer Science senior at New York University Shanghai. She has worked in art, media art, and software, and harbors a untapped love for astronomy and physics.
Siyuan Yu is a Computer Science senior at New York University Shanghai. He has worked extensively in game design and development, and holds a love for physics and chemistry.

Acknowledgements

Thank you to our supervisor Tim Byrnes for guiding us into the world of quantum computing and for his patience with our questions. Thank you to Olivier Marin for the guidance for building our capstone research and moral support during these unprecedented times in the Shanghai lockdown.
Robin Luo would like to thank her lockdown friends who flash delivered her a Mac charger in mid-April to complete this capstone, Albert Luo and Sherry Huang for supporting her along her academic journey, and Yanyi and Robert who have been there for her, always.
Siyuan Yu would like to thank Yumo Yao, Shih-Huan Chou, Mengjie Shen and Xiao Wang for the mental and material support during the hard times of 2022 Shanghai lockdown.

Abstract

Our work shows how to show and analyze quantum supremacy algorithms given random circuit noise. Random circuit noise makes quantum algorithms less reliable and stable. Using IBM Q and Qiskit to generate random quantum circuits, we conducted a complexity analysis on our algorithm by sampling from a random quantum circuit and showing an entropy of $\approx$ 3.5 at a circuit depth of $\approx$ 50. By showing entropy, we can show the randomness of our circuit, providing the foundation for demonstrating quantum supremacy.

Keywords

Quantum Supremacy; Random Quantum Circuits; Entropy; Quantum Computing; Random Circuit Sampling; Cross-Entropy Benchmarking

Contents

1 Introduction

In recent years, quantum computing has made great leaps in its promise to offer computational speed by using quantum phenomenon and technology to break the bounds of classical computation. [1] Despite these exciting new advances, quantum computing has its limitations – as John Preskill famously in 2018 coined the term to describe this era of quantum computing, NISQ, or the *Noisy Intermediate Scale Quantum.* NISQ era describes how quantum computers now will be able to handle intermediate executions of 50-100 qubits which will surpass classical computing abilities but still suffer from quantum gate noise which will limit circuit size and cause unreliable algorithmic performance. Because of these limitations, fault-tolerant quantum computing is still a prospect that remains out of our grasp. [2]

In 2019, Google's AI Quantum Computing Lab made groundbreaking research by experimentally showing the quantum supremacy – a quantum computer can run algorithms no classical computer can – is achievable. They came up with a quantum processor that samples one instance of a quantum circuit a million times in 200 seconds, which greatly surpasses performance of a state-of-the-art classical supercomputer running time for the equivalent task would take, which is approximately 10,000 years.[3]

After the demonstration of quantum supremacy, the quantum algorithm studies have been promoted globally. Some well-known quantum algorithms have been studied and implemented with Qiskit, an open source SDK working with OpenQASM, and can be run in IBM Q quantum processors.

Improvements towards quantum algorithms, quantum circuits, and qubit quality are large motivators in quantum computing research in NISQ era. [2] Recent developments show that the quantum algorithms are now possible with the current processors in quantum computing, and can surpass the classical computing in speed. Thus, for our research, we would like to show the supremacy of quantum computing similar with Google's research in 2019, but with our own randomly generated quantum circuits. We will use random circuit sampling to sample from the generated random quantum circuit's output distrbutions. Then we will examine its entropy of the random circuit to compare it to the ideal values with IBM Q and Qiskit in a process called cross-entropy benchmarking in order to show the quantum supremacy.

2 Related Work

2.1 Introduction

This section we will introduce the related work to show quantum supremacy in the field of quantum computing. We will look at fundamental components of random quantum circuits, qubits, and quantum gates. Then, we will explain various quantum measurements such as entropy and fidelity, and old and new benchmarking techniques used in the field. We will then briefly overview the research to approach demonstrating quantum supremacy and the tools about IBM Q and Qiskit for simulating quantum computing in classical computers.

2.2 Qubits

Similar to the classical computer that uses bits to process computation, quantum computers uses qubits. However, different from bits, that has two certain states 0 and 1, the states of qubits or quantum bits, can be in a coherent superposition of the basis states. [1] This means that a single qubit can be described by a linear combination of $|0\rangle$ and $|1\rangle$.

However, the quantum qubits noises can cause decoherence, which essentially drives the qubits to unentangled states. Thus, one way to improve this decoherence is to apply random quantum circuits, instead of the well-designed biased circuits.

2.3 Random Quantum Circuits

A random quantum circuit describes a quantum algorithm where at every cycle of the circuit is a randomized set of single-qubit gates and a multi-qubit gate. The total cycles in the quantum circuit represent its circuit depth. Random quantum circuits are used in modern benchmarking techniques to demonstrate quantum supremacy, randomized circuit sampling, and cross-entropy benchmarking [3, 4, 5].

2.3.1 Quantum Gates

A quantum logic gate (or simply quantum gate) is a fundamental element of a quantum circuit that operates on a small number of qubits in quantum computing and specifically the quantum circuit model of computation. They serve as the foundation for quantum circuits, much as classical logic gates do for regular digital circuits.[6] Unlike many classical logic gates, quantum gates with exception of the measurement and reset options, are reversible, which means the input can be deduced from the output. Quantum gates are unitary operators, thus researchers uses unitary metrics to represent the quantum gates. The following quantum gates and its representation is given from *Quantum Gates* [6].

The most basic gate is the identity gate. The identity gate is represented by the identity matrix. It operates similar to the wires in the classical logic gates, and is usually written as I, and is defined for a single qubit as the metrics as

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

I is basis independent and does not modify the quantum state.

2.3.2 Single-qubit gates

Notable single-qubit gates are the Pauli gates and the Hadamard gates.

Pauli-X Gates is a gate that does similar operations with the NOT Gates in classical logic gates, which transfers $|0\rangle$ into $|1\rangle$ and transfers $|1\rangle$ into $|0\rangle$. It is represented as

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pauli-Y Gates and Pauli-Z Gates are similar to the Pauli-X Gates. However, Pauli-Y Gates transfers $|1\rangle$ into $i|0\rangle$ and transfers $|0\rangle$ into $-i|1\rangle$, and is represented as

$$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Pauli-Z Gates holds $|0\rangle$ unchanged and changes $|1\rangle$ into $-|1\rangle$ and is represented as

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Hadamard gates are fundamental quantum gates because it creates superpositions of given basis states. It transfers a $|0\rangle$ into $\dfrac{|0\rangle + |1\rangle}{\sqrt{2}}$ and transfers $|1\rangle$ into $\dfrac{|0\rangle - |1\rangle}{\sqrt{2}}$. The Hadamard Gates itself is represented as

$$\mathrm{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

2.3.3 Multi-qubit gate: CNOT

Controlled gates or CNOT gates operate on two or more qubits, with one or more qubits serving as a control for a specific operation and the other qubit as a target. CNOT gates will hold the controlled qubit and operate on the target qubit. It holds the state for the control qubit, and also hold the state for target qubit if the control's state is $|0\rangle$ and transfers target's state from $|0\rangle$ into $|1\rangle$ or from $|1\rangle$ to $|0\rangle$ if the control's state is $|1\rangle$. CNOT gate is represented by the matrix as

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

A random quantum circuit needs to randomly produce n qubit-states and be of sufficient circuit depth, that are picked uniformly from all possible states [7, 3]. To have a random circuit produce n qubit-states, single and multi-qubit gates should be used to produce a sufficient output distribution when sampling and measuring random quantum circuits.

2.4 Quantum Measurements

2.4.1 Entropy

The notion of entropy was introduced by Clasuius in order to describe the disorder of thermal behavior, but it is used as a quantity expression in quantum computing to show quantum uncertainty. Shannon noticed the importance of the information carried in the randomness or uncertainty of a physical system. Thus Shannon came up with the equation calculate Shannon entropy as below

$$H(X) = E[I(X)] = E[-log(P(X)] \qquad [8]$$

This calculation is not consistent when it comes to quantum physics where probability is describe as Density Matrix instead of the single probabilities. For quantum cases, the Von Neumann Entropy is more widely applied to describe the randomness of quantum sytem with the formula

$$S = -tr(\rho * \ln \rho). \text{ [9].}$$

where tr for the trace of matrix to, which is the sum of elements on the main diagonal (from the upper left to the lower right) and ρ is a probability of a bitstring exhibited by the quantum circuit.

Or alternatively as

$$E = -\sum_k (p_k * \ln p_k). \ [4].$$

where k represents number of probabilities in the output distribution given by a particular measurement outcome of the quantum circuit, and p_k is a probability of the bitstring exhibited.

The circuit's entropy is needed in order to demonstrate quantum supremacy [10, 11, 4, 12]. In practice, to observe the circuit's entropy at an enough depth, circuits will exhibit a convergence towards a Porter-Thomas probability distribution for the measured probabilities, a characteristic to show quantum chaos [10]. This measurement is a key step in cross-entropy difference which is then used in cross-entropy benchmarking to measure the algorithm's performance and confirm quantum supremacy [12, 11, 10].

2.4.2 Fidelity

In addition to understand a random circuit's functionalities, a fidelity is introduced to characterize these distortions from quantum noise and the state of qubits, which is a measurement of the distance between two quantum states, $|0\rangle$ and $|1\rangle$. [1] Fidelity estimations are significant to measure errors and observe the circuits are functioning correctly and are used in techniques such as randomized benchmarking (RB) and cross entropy benchmarking (XEB) to analyze quantum circuits [13, 5, 10].

2.4.3 Benchmarking

Randomized benchmarking is a popular benchmarking tool for quantum devices with a small number of qubits to measure quantum noise; however, research has shown that RB is not scalable as current hardware sizes increase because of the circuit gates it uses, called Clifford gates. When scaled, "the state of the system quickly converges to maximally mixed state before meaningful information can be extracted" and thus a standard RB is only limited to 3 qubits, but other modifications of this benchmarking technique have shown to handle a mere 10-20 qubits. [13]

Randomized benchmarking relies on a small group of gates called Clifford gates; however, a new technique to show complexity theoretic properties, randomized circuit sampling or RCS are different in two main aspects: the flexibility in circuit gates and its more stable system state. It can "efficiently extract the total amount of quantum noise of a many qubit system by creating an exponential decay of fidelity" and meaningful information can still be extracted from it [13, 5]. Random circuit sampling describes a task to "take a quantum circuit of a specific form, in which each gate is chosen randomly, and generate samples from its output distribution" [5]. Unlike randomized benchmarking, it is much easier to scale its circuit form using the RCS method since all that is needed is to add a cycle of randomized gates to the end of a circuit to increase its depth.

Given a sampled random quantum circuit, cross-entropy benchmarking or XEB is used in conjunction, which is a method used to estimate fidelity and validate them in more complex multi-qubit circuits. The difference, known as the cross-entropy difference or XED, found in cross-entropy benchmarking is "a measure of correspondence between experimentally obtained samples and the output distribution of the ideal circuit", obtained through sampling from the Porter-Thomas distribution [10]. In order words, through XEB, we are able to analyze the quantum algorithm's performance by comparing the ideal circuit entropy to our measured circuit's entropy in XED [12, 10]. Through these methods of RCS and XEB with a direct simulation of random quantum circuits can a near-term quantum supremacy be realized. [13, 10, 3]

2.5 Quantum Supremacy Approaches

Frameworks for demonstrating quantum supremacy all rely the common computational task of sampling from an output distribution from a quantum algorithm. To establish quantum supremacy, the quantum algorithm's hardness and verification must be shown. Hardness means no classical algorithm can sample from the output distribution close to the quantum algorithm's, and verification means to show the robustness of its difficulty to sample from its output distribution [5].

In 2019, Google provided a demonstration of quantum supremacy through designing a high-fidelity 56 qubit quantum processor called Sycamore. This processor can achieve high-fidelity in one- and two- qubit operations, meaning a significant improvement in qubit and gate error through the creation of more fault-tolerant circuits. In their research, they "[sampled] the output of a pseudo-random quantum circuit" by having using random circuits to produce a set of bitstrings like $\{0000101, 1011100, \dots\}$ [3]. Their circuit pattern is designed by randomly choosing single-qubit gates that do not repeat sequentially and sequentially tiling two-qubit gates to its four nearest neighbors in order to make certain bitstrings more likely to create a probability distribution that mimics the pattern formed in quantum interference as famously shown in the notable quantum physics's double slit experiment. [3]

Using their processor, Google used both the popularized benchmarking techniques of random circuit sampling to quantify gate errors and measure fidelity and cross-entropy benchmarking to verify the hardness of the quantum operations and demonstrate quantum supremacy.

2.6 Quantum Algorithms with IBM Q and Qiskit

Only for the past few years driven by quantum hype, the practical uses of quantum computing have been available. Aside from Google's quantum computing research, IBM is also another company which has been active in quantum research. Through a cloud interface platform, IBM has made the largest quantum computer, IBM Quantum, shortened as IBM Q, accessible for public use, allowing users to execute operations with self-drawn circuits and programs made with programming languages like the open source OpenQASM SDK, Qiskit. With the IBM Q, different small n number qubit processors are available for use and have shown up occasionally in quantum algorithm research [14, 15]

2.7 Conclusion

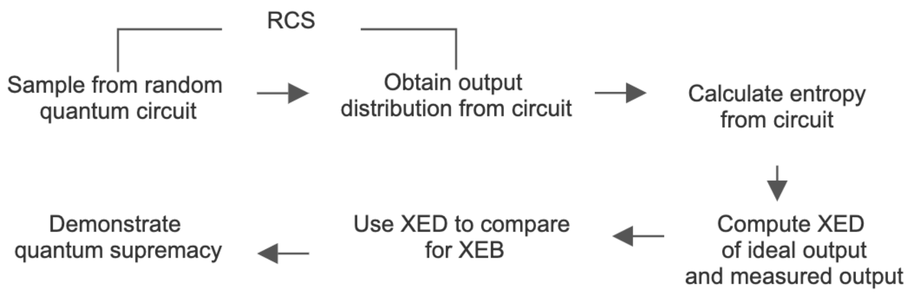
RCS
Sample from random quantum circuit
Obtain output distribution from circuit
Calculate entropy from circuit
Compute XED of ideal output and measured output
Use XED to compare for XEB
Demonstrate quantum supremacy

Figure 1: A generalized chart for demonstrating quantum supremacy. RCS stands for Random Circuit Sampling, XED stands for cross-entropy difference, and XEB stands for cross-entropy benchmarking

To understand how to do a complexity and demonstration of quantum supremacy from a quantum algorithm, we must understand what is a random quantum circuit and how it is built. Random quantum circuits are made up of quantum bits or qubits that contain a state information, and random quantum gates where each gate does a different operation to the qubits' states. Single and multi-qubit gates are used to form a quantum circuit that can run a quantum program or quantum algorithm. In order to demonstrate quantum supremacy, we need to show the hardness and verification of a quantum algorithm from a quantum circuit. Modern methods apply random quantum circuits and increase its circuit depth, meaning add cycles of single and multi-qubit gates, and then sample from its output distributions. This complexity-theoretic process is called random circuit sampling. Next, entropy is calculated from the output distribution to observe its convergence to a Porter-Thomas distribution. This property can be used in calculating the ideal output against our measured output of our random quantum circuit through utilizing cross-entropy difference and cross-entropy benchmarking, so that it is possible to demonstrate quantum supremacy. This process of RCS and XEB is widely used in demonstrating quantum supremacy in the quantum computing field, as seen as in Google's notable research. We will show an implementation of this process in our own random quantum circuit and quantum algorithm through Qiskit and IBM Q, as little research is done using these frameworks and services.

3 Solution

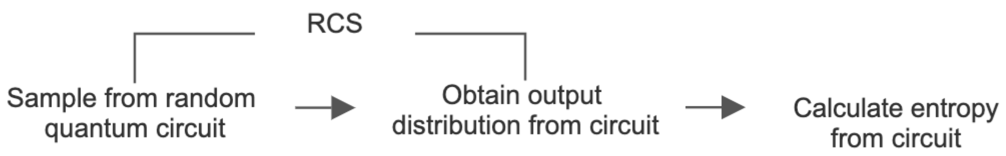
RCS
Sample from random quantum circuit
Obtain output distribution from circuit
Calculate entropy from circuit

Figure 2: A generalization of our procedure taken for our current solution. RCS stands for random circuit sampling. Entropy refers to the Von Neumann entropy.

This section will discuss the architecture, methodology, and findings of our objectives that we managed to reach in order to do a complexity and demonstration of quantum supremacy algorithms with Qiskit and IBM Q.

3.1 Architecture

The architecture we used included IBM Quantum, a cloud service which offers access to a variety of quantum systems, Qiskit, an OpenQASM SDK using the Python programming language used for creating quantum circuits, and Jupyter Notebook to run the program simulation.

Qiskit provides a library which users can connect IBM Q to the Python program written with

Qiskit imported modules. It can run quantum circuits on Qiskit Aer module with a simulated IBM Q backend and also run them on a real quantum computer like IBM Q. Using Python, we programmed a random quantum circuit and ran it to completion on the Qiskit Aer's QasmSimulator.

3.2 Procedure

By using the Jupyter notebook environment and Qiskit, we generated 41 5-qubit random quantum circuits where each circuit added an additional 5 cycles to the end of the previous generated circuit to reach a total circuit depth of 201 or 201 cycles as shown in Figure 3. Each cycle we randomly generated for our random quantum circuit was a series of Pauli-X, Y, Z gates, along with Hadamard gates where each generated gate were to act on a single qubit. These single-qubit gates were then separated with a CNOT gate acting on a set of two randomly chosen qubits on the circuit that were to signify a cycle.

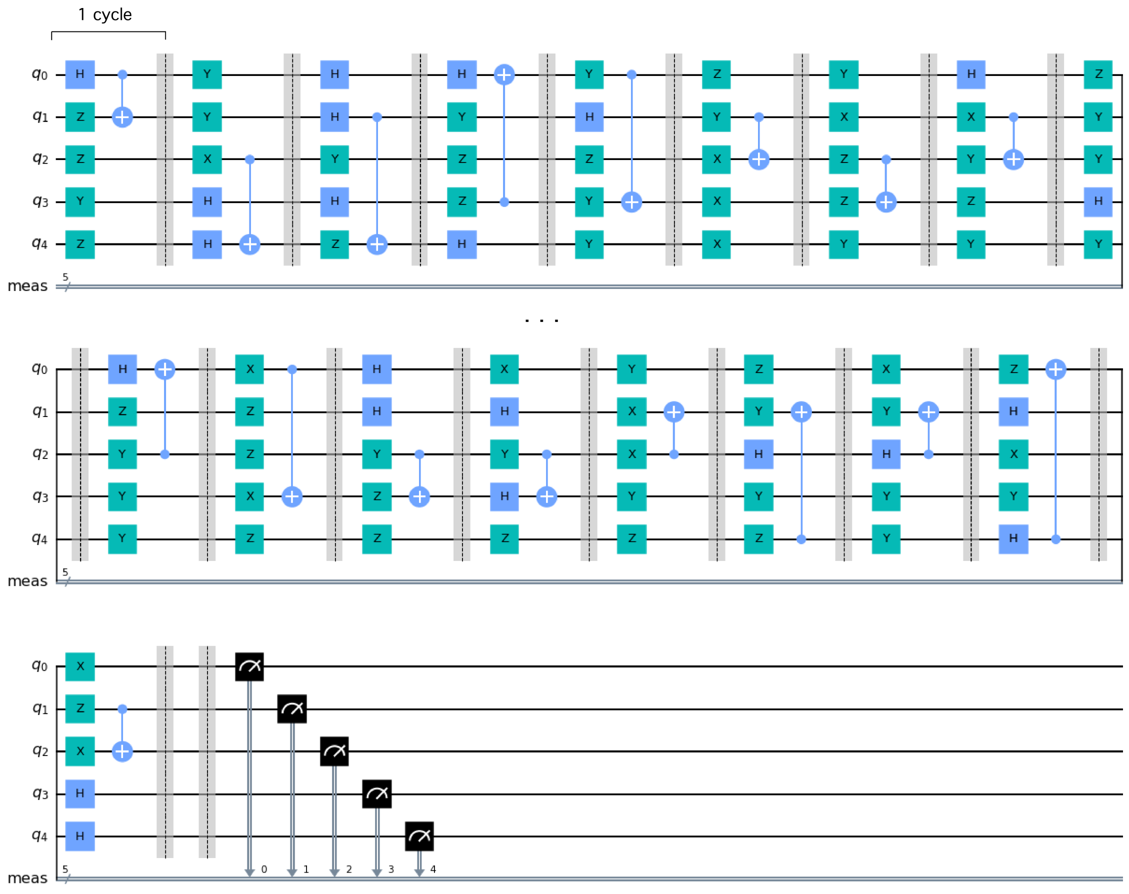

Figure 3: Random quantum circuit with circuit depth of 201 (L = 201) generated with Python and Qiskit

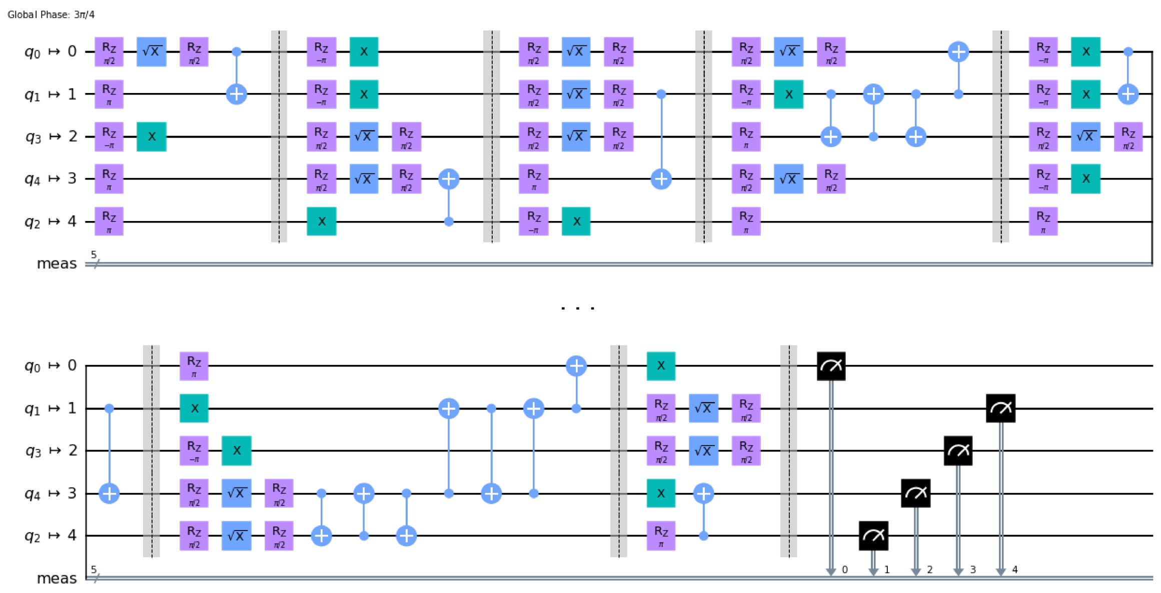

Figure 4: Transpiled random quantum circuit generated from IBM Q's backend with circuit depth of 201 (L = 201) as shown in Figure 3

Each of the 41 circuits were measured on all 5 of its qubits and then the circuit was transpiled with IBM Q's backend as shown in our random quantum circuit of a 201 circuit depth in Figure 4 below as an example. The output distributions for the transpiled circuits were extracted by simulating each circuit on Qiskit's QASMSimulator, then used to find each circuit entropies at the incremented circuit depths. The overall behavior of our generated random quantum circuit was observed in our findings.

3.3 Findings

We found with our generated random quantum circuit that its Von Neumann entropy converged to $E \approx 3.5$ at a circuit depth of $L \approx 50$ with the Qiskit QASM Simulator with a simulated IBM Q backend.

Figure 5 below is the graph of each of the entropies we obtained from our random quantum circuit from 1 then at an increment of every 5 cycles up to 201. E represents the Von Neumann entropy. L represents the number of cycles or circuit depth.

By showing a total entropy of $E \approx 3.5$ at a point of a circuit depth of $L \approx 50$, we show the complexity and randomness of our random quantum circuit's quantum algorithm, providing groundwork to demonstrate quantum supremacy for our algorithm.

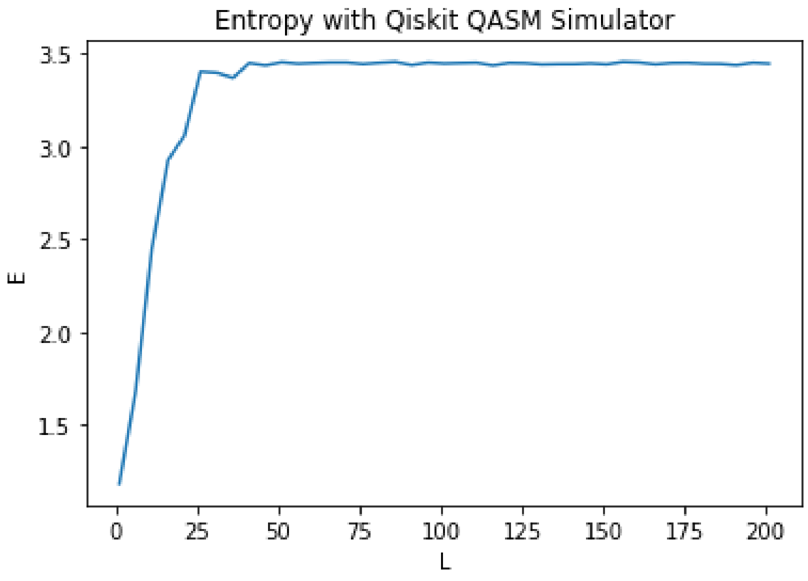

Figure 5: Entropy graph on Qiskit's QASM Simulator of our transpiled random quantum circuit up to a circuit depth of 201 in Figure 4

4 Results and Discussion

4.1 Experimentation protocol

The detailed steps of our experimentation protocol which brought us to our solution is as follows. The GitHub repository of the Python program is linked here, and the Jupyter Notebook is linked here.

We used the Python programming language and Qiskit modules to first simulate random quantum circuits of a single circuit depth or a single cycle of randomized single-qubit Pauli-X, Y, Z, and Hadamard gates, and a CNOT gate that acted on a randomized set of two qubits.

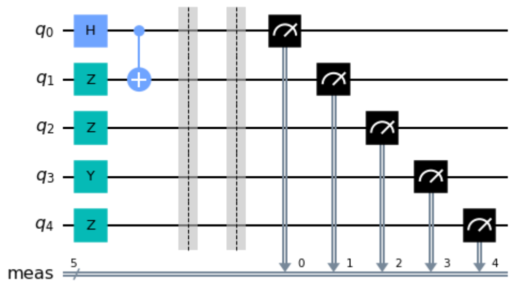

Figure 6: Random Quantum Circuit with circuit depth of 1 (L = 1)

Figure 6 shows the first iteration of our random quantum circuit with a circuit depth of 1.

The circuit was first generated by creating a random quantum circuit of 5-qubits with no gates, then a single cycle was added. In order to add a cycle, the circuit is converted to a OpenQASM string. Gates and its positions on the circuit's qubits are randomly generated and added to OpenQASM string in the appropriate string format in order to convert it into an instruction to recreate the circuit. Each cycle is added by concatenating the randomized X, Y, Z, H gates, and a CNOT gate in OpenQASM string to the previously generated circuit's OpenQASM string instructions.

New instances of the random circuit were created by adding 5 cycles to the end of its circuit. This process of adding 5 cycles to the end of the random circuit was repeated an another 40 times from the random circuit of a circuit depth of 1 as we created in Figure 6, for a total of 41 instances of our random quantum circuit at a particular circuit depth.

We stored 41 random quantum circuits generated into a list. The first iteration of our random circuit was of 1 cycle, and the last was of 201 cycles. The final iteration of our random quantum circuit generated was of 201, as shown in Figure 3.

Each 41 circuits generated were stored, measured, and then transpiled with the simulated IBM Q backend from Qiskit. Figure 7 is a transpiled circuit of Figure 6 of the Qiskit generated random quantum circuit of a circuit depth of 1. Likewise, we transpiled each of the remaining 40 circuits until we transpiled the final circuit of a depth of 201 as seen in previously in Figure 4.

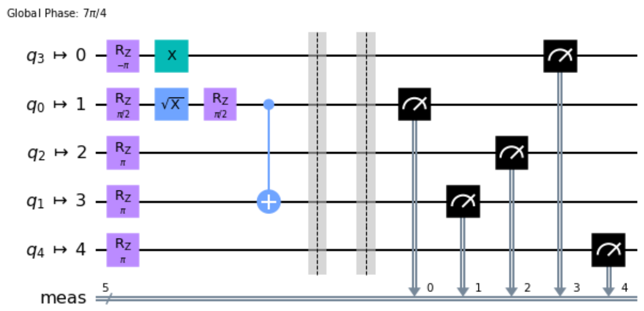

Figure 7: Transpiled random quantum circuit generated from IBM Q's backend with a circuit depth of 1 (L = 1) as shown in Figure 6

Each of the circuits we transpiled then executed a job at given number of shots. Every job produced results which gave the counts of the produced bitstrings from the circuit. The number of shots we used for the job was 1024. The counts were returned from the results as a dictionary that stored each time each bitstring appeared during the job. Figure 8, Figure 9, and 10 show the output distributions of each job done on the circuits at a particular circuit depth of 1, 26, 56, 101, and 201.

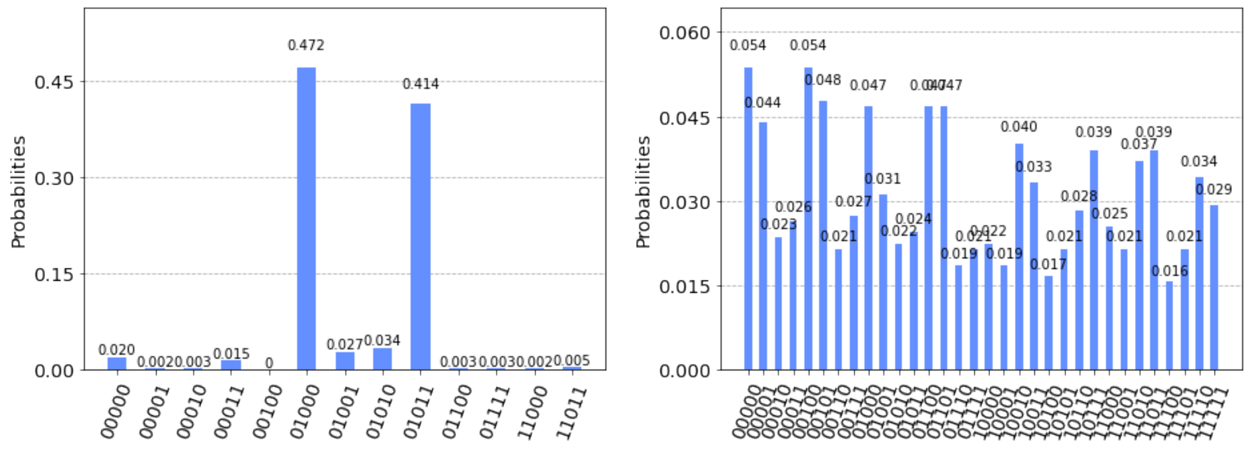

Figure 8: Output distribution of our random quantum circuit on the left at circuit depth of 1 ($L = 1$) and on the right at circuit depth of 26 ($L = 26$) on Qiskit's QASM Simulator

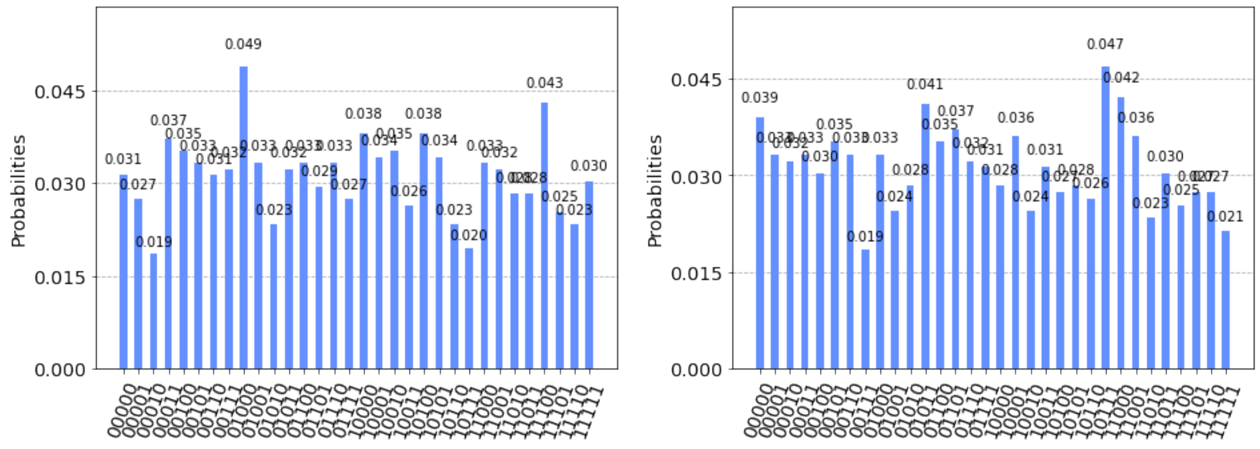

Figure 9: Output distribution of our random quantum circuit on the left at circuit depth of 56 ($L = 56$) and on the right at circuit depth of 101 ($L = 101$) on Qiskit's QASM Simulator

The output distribution of the random circuit at a certain circuit depth was then stored in order to calculate the overall entropy. In order to do so, we first calculated the probabilities obtained from the job through this equation as follows

$$p_k = \frac{count}{shots}$$

where p_k describes a sample probability of a bitstring occurring in the random quantum circuit, count describes the number of times a bitstring occurs in the measurement outcome of the simulated job, and shots represent the amount of times the job was executed, which was 1024 for our experiment.

We used the following representation of the Von Neumann entropy formula to calculate the entropy of each transpiled circuit

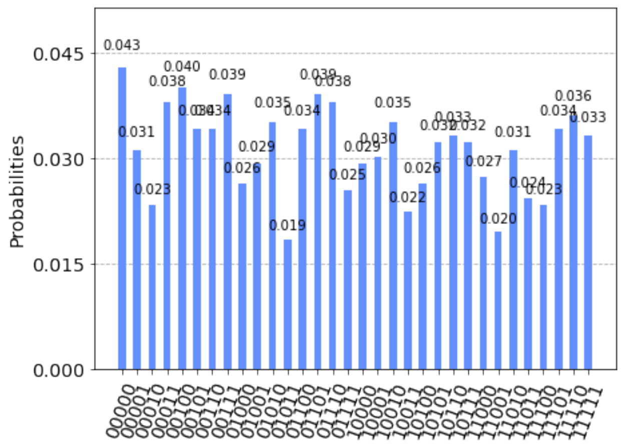

Figure 10: Output distribution of our random quantum circuit at circuit depth of 201 (L = 201) on Qiskit's QASM Simulator

$$E = -\sum_k (p_k * \ln p_k) \ [4].$$

where k represents number of probabilities in the output distribution given by a particular measurement outcome of the random quantum circuit, and p_k is a probability of the bitstring exhibited.

The entropy of the random quantum circuit at the particular cycle was then stored into a list to observe the overall behavior of our random quantum circuit as we initially created in Figure 3. The result received is shown in our graph in Figure 5 where we observe $E \approx 3.5$ at $L \approx 50$.

4.2 Protocol Discussion

For our solution, we managed to run a simulation of our random quantum circuit on the Qiskit Aer's QASMSimulator with a simulated IBM Q backend. An additional step we took was simulating our random quantum circuit on a real IBM Q processor, Quito, a 5-qubit QV16 superconducting processor. However, running an instance of our random quantum circuit on the actual IBM Q took a considerable amount of time. We still managed to produce a real output distribution using Quito with our transpiled random quantum circuit at a circuit depth of 56 as shown in Figure 11, near the range of what was simulated where the entropy converged as observed on the Qiskit's QASMSimulator.

The entropy that was calculated from the real output distribution was of $E \approx 3.4 \approx 3.5$ at $L = 56$, as expected from the simulated results we produced.

Referring to the generalized process of quantum supremacy algorithms in Figure 1, we only managed to complete about half of the process due to the time constraints and unforeseen cir-

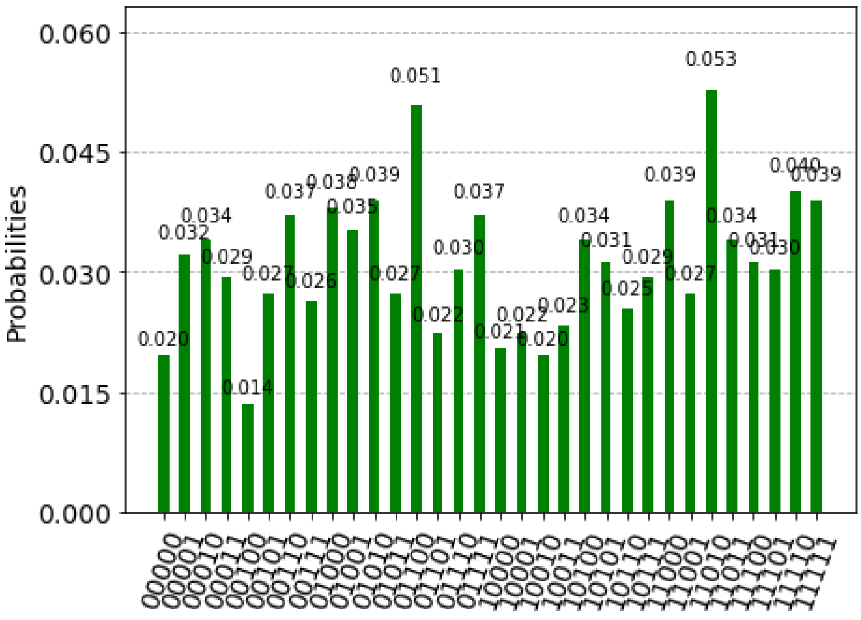

Figure 11: Real output distribution on IBM Q's Quito of our random quantum circuit at circuit depth of 56 (L = 56)

cumstances of the capstone project.

The next steps for this capstone include calculating the overall real entropy of our random quantum circuit on the IBM Q's Quito, then observing its convergence to the Porter-Thomas distribution to receive the XED of the ideal output and the real measured output from IBM-Q. Then, we will be able to do XEB in order to demonstrate quantum supremacy through verification of the hardness of our quantum algorithm using IBM Q and Qiskit.

5 Discussion

The overall process of this capstone was divided into two parts, the learning and understanding the basics of quantum computing, and experimenting and practicing with quantum circuits and the quantum algorithms with Qiskit and IBM Q. In this section, we will address the challenges in areas where we had difficulty in concepts, unsuccessful directions taken which helped us lead to our final results, and the suggested advice for the next steps.

5.1 Finding Direction with Quantum Computing

We had many iterations of a direction towards a solution for our capstone that were removed in the final solution. However, these discarded iterations were crucial in forming a better understanding of what our objectives meant.

One of our challenges was understanding the linear algebra which formed the basis of quantum computing. The Quantum Computing lectures provided by our supervisor and *Quantum computation and quantum information* provided basics in understanding complex numbers, vector spaces, and matrix terminology abstracted for quantum computing; however, becoming familiar with this terminology was difficult in conjunction to understanding how to show quantum supremacy. While we introduce basic linear algebra and its relationship to quantum gates, our limiting knowledge made it difficult to understand the more complex topics and the mathematical jargon of such concepts like Porter-Thomas distribution, entropy, XED, and XEB commonly seen in the research we did. While our research may not follow these concepts in depth, familiarizing with these topics still were crucial in informing us the basic idea of demonstrating quantum supremacy. We advise to spend more time on the mathematics in order to easily understand the context of the field's research.

Understanding the relationship of quantum algorithms and random quantum circuits was another challenge. Our understanding of complexity comes from analyzing it in classical computers, often in high-level programming form, rather than decomposing the algorithm down into low-level bits. Quantum computing algorithms analyze complexity by looking in this low-level manner by looking at the generated qubit strings and using statistics. Reorienting our understanding of what complexity meant was a challenge for us, as we often ran into difficulty understanding what it meant to analyze the complexity of a random quantum circuit.

Thus, in our experiments, we ran into difficulties in understanding what "random" meant in a quantum circuit and its connection to complexity. We were not clear how Google's random quantum circuits was still "random" and used RCS when it was "pseudo-random." Since we were most familiar with Google's paper, a direction we took was attempting to recreate Google's random circuit as an experimental direction to demonstrate quantum supremacy (see 5.2.2 for more). Because of this confusion about random circuits, we also did not see the connection to quantum complexity. We spent time exploring various algorithms and complexity related to quantum computing such as query complexity, Shor's Algorithm, and Grover's algorithm to help us connect these concepts. However, the time spent learning these concepts were not entirely applicable to our final solution.

Reaching out to our supervisors proved to be a crucial anchor in showing us toward the correct paths to take and clarifying our confusion in the various terminology we had little knowledge of.

We realized reaching out to our supervisors even over the most basic confusions helped us clarify the steps needed to take for our objectives much faster, in particular drawing the connections between random quantum circuits, RCS, complexity, entropy, XEB, and quantum supremacy.

Considering the capstone had a tight time constraint, we recommend to reach out more often next time to maximize the time spent on the experimental portion, as well as ask more direct and clarifying questions about the researched topics at hand. For example, why or why not they are applicable to our objectives, what is the purpose of doing such a process, and what does a concept mean and achieve.

During this process, we learned how to position ourselves within the research, what we did not know, and what we had learned. We developed an important practice to guide us toward better questions and answers to enrich our research. We also learned the steep learning curve in our capstone topic, yet because of choosing a topic we never encountered before, we learned and reflected how to be better at synthesizing information and researching.

5.2 Finding Direction with Qiskit and IBM Q

5.2.1 Hello, World

We used the "Hello World" example to get started on Qiskit. Note, computers running on the Windows Operating System may run into errors when downloading the Qiskit library. Packages "pillow" and "pylatex" may need to be downloaded separately if users want to draw the circuits on Anaconda.

5.2.2 Generating Random Quantum Circuits with Qiskit and IBM Q

Qiskit Documentation showed us a random circuit function that can automatically create a circuit at a certain number of qubits and at a certain circuit depth. We created a simple simulation in Jupyter Notebook using the available random circuit function, then attempted to recreate Google's quantum circuit layout. By doing so, we looked at the Qiskit GitHub to analyze the library at a Python programming level in order to make more complex adjustments in mimicking the circuit layout. We copied the random circuit function from its GitHub repository, and made minor adjustments in the code. Because we were copying the layout of Google's circuit to demonstrate quantum complexity, we first looked at reproducing the $\sqrt{X}$, $\sqrt{Y}$, $\sqrt{W}$ gates in Qiskit. Despite finding the $\sqrt{X}$ gate, the $\sqrt{Y}$ gate was not available, and this gate is also needed to recreate the $\sqrt{W}$ gate (see **2.4**). We decided creating the $\sqrt{Y}$ gate could be a possible next step to give us an idea of how to approach random circuits.

However, after discussion with our supervisor, we learned it was not necessary to do so. We were referred to look at Google's paper again, as well as the random circuit analysis procedure listed in "Quantum supremacy with spin squeezed atomic ensembles." We understood from our discussion that we were to create a random circuit by producing a random circuit layout and changing the gates used during each job we ran in order to produce probabilities to find an entropy. We discarded the previous code and returned to our original simple simulation created by the random circuit function. By further exploring Qiskit's documentation and its GitHub, we found a function which could generate a OpenQASM string of instructions which described the circuit layout. By using this function then replacing the string's gate names and converting the OpenQASM string back to an instruction to produce a circuit layout, we developed a method to create a new quantum circuit layout by simple string manipulation.

We referred to Qiskit's Textbook and Circuit Basics to help us get started on adding quantum gates to our circuit and generating measurements from a circuit in order to find the probabilities of qubit states. All of these challenges and its remaining piece helped us piece together the final solution and its connections to our research.

5.3 Next Steps

For our solution, we managed to demonstrate the entropy of our random quantum circuit we randomly generated with Python, Qiskit, and IBM Q. We advise for next steps to finish generating the real entropy with IBM Q's Quito. We recommend to first run the simulation on Qiskit's simulated IBM Q backend, and then run it on the real IBM Q processor to the simulated estimated entropy's circuit depth and lessen the amount of random circuit instances to reduce the waiting time to execute all the real jobs.

Then, we advise to consider the meaning of the entropy, and to understand why without multi-qubit gates or the CNOT in our case, the entropy is said to have none. We did some research in exploring multi-qubit gates and entanglement, and we imply that there is a relationship between the entropy and entangled qubits that may be worth exploring.

Other additional would be to research more in depth of Porter-Thomas distribution and what does that mean in the quantum computing field, how it characterizes quantum chaos, and is used as an ideal marker for demonstrating an ideal circuit. We would advise to clarify P-T distribution relationship with XED, as well as XED in XEB asides from it comparing with an ideal circuit. We believe with more mathematics and quantum computing background, these conceptual relationships would be much more clearer.

Our capstone provides the groundwork for which further research is possible for the random quantum circuit we developed. In addition to laying the groundwork to demonstrating quantum supremacy on our random quantum circuit, we also developed a tool that easily generates random quantum circuits with popular quantum computing services IBM Q and Qiskit. Our experimentation makes it easier to use RCS on a random quantum algorithm of our layout using X, Y, Z, H, and CNOT gates, as well as allow room for a possible implementation for other gates and manipulations of different circuit layouts in the future.

6 Conclusion

Our objectives for our capstone was to show complexity and demonstration of quantum supremacy algorithms through IBM Q and Qiskit. In order to show it, we must perform random circuit sampling onto a random quantum circuit we generated and then cross-entropy benchmarking with our simulated code in Qiskit and IBM Q.

We managed to calculate a Von Neumann entropy from sampling our our random quantum circuit generated with Python and Qiskit. We simulated the code on Qiskit Aer's QASMSimulator by using transpiled circuits with a simulated IBM Q backend. In the end, 41 instances of our random quantum circuit of a circuit depth from 1 to 201 was produced with each instance adding 5 cycles to the end of the previously generated instance of the circuit. We used Pauli-X, Y, Z, and Hadamard gates for our single qubit gates and CNOT for our multi-qubit gates to signify a cycle for our random quantum circuit.

After doing random circuit sampling, our generated random quantum circuit was found to converge to an entropy of $E \approx 3.5$ at around the total circuit depth of $L \approx 50$.

Additionally, we began to perform a real entropy analysis by running our random quantum circuit on the IBM Q's Quito 5-qubit superconducting processor; however, due to the time constraints and the waiting time to generate the job results, we only managed to only test one instance of our random circuit at a particular circuit depth. We tested the instance of our random circuit at $L \approx 56$ where its entropy was shown to be as expected, $E \approx 3.5$.

For the next steps, we recommend to finish analyzing overall entropy of our random quantum circuit with Quito, to observe its convergence to Porter-Thomas distribution, and finally to do cross-entropy benchmarking in order to finish doing a possible demonstration of quantum supremacy using IBM Q and Qiskit.

References

[1] M. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2018.

[2] J. Preskill, “Quantum computing in the nisq era and beyond,” 2018.

[3] R. B. e. a. Frank Arute, Kunal Arya, “Quantum supremacy using a programmable superconducting processor,” no. 574, p. 505–510, 2019.

[4] T. B. Yuheng Shi, Junheng Shi, “Quantum supremacy with spin squeezed atomic ensembles,” 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2204.11772

[5] C. N. U. V. Adam Bouland, Bill Fefferman, “On the complexity and verification of quantum random circuit sampling,” vol. 15, p. 59–163, 2019. [Online]. Available: https://doi.org/10.1038/s41567-018-0318-2

[6] C. Williams, “Quantum gates,” 2011. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-1-84628-887-6.pdf

[7] N. H.-J. Dalzell, Alexander M. and F. G. Brandao., “Random quantum circuits anti-concentrate in log depth,” 2020. [Online]. Available: https://arxiv.org/pdf/2011.12277.pdf

[8] D. M.Ohya, *Quantum Entropy and Its Use*, 2nd ed. New York: Springer-Verlag Berlin Heidelberg New York, 2004, ch. 1.

[9] S. Kak1, “Quantum information and entropy,” 2007. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10773-006-9245-6.pdf

[10] I. S. S.-V. e. a. Boixo, S., “Characterizing quantum supremacy in near-term devices,” vol. 14, p. 595–600, 2018. [Online]. Available: https://doi.org/10.1038/s41567-018-0124-x

[11] R. B. e. a. Frank Arute, Kunal Arya, “Supplementary information: Quantum supremacy using a programmable superconducting processor,” no. 574, 2019. [Online]. Available: https://static-content.springer.com/esm/art%3A10.1038%2Fs41586-019-1666-5/MediaObjects/41586_2019_1666_MOESM1_ESM.pdf

[12] S. Mullane, “Sampling random quantum circuits: a pedestrian’s guide,” *Quantum Physics*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2007.0787

[13] R. B. L. J. B. F. Yunchao Liu, Matthew Otten, “Benchmarking near-term quantum computers via random circuit sampling,” 2021. [Online]. Available: https://arxiv.org/pdf/2105.05232.pdf

[14] M. E. DeCusatis, C., “Near term implementation of shor’s algorithm using qiskit,” in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2021, pp. 1564–1568. [Online]. Available: https://doi.org/10.1109/CCWC51732.2021.9376169

[15] U. Skosana and M. Tame, “Demonstration of shor’s factoring algorithm for n=21 on ibm quantum processors,” vol. 14, p. 595–600, 2018. [Online]. Available: https://doi.org/10.1038/s41567-018-0124-x